IN THE CLAIMS:

Claims 2-27 are added. All pending claims and their present status are produced below.

1. (Original) A method for performing arithmetic in a memory to memory architecture in an embedded processor, the method comprising:

receiving a 32-bit fixed length instruction, the instruction specifying a source address in a memory using 11 bits, a source address in a register file using 5 bits, a destination address in the memory using 11 bits, and a mathematical operation to be performed using 5 bits; and

responsive to receiving the fixed length instruction:

accessing, from the source address in the memory, a first operand on which the mathematical operation is to be performed;

accessing, from the source address in the register file, a second operand on which the mathematical operation is to be performed;

performing the mathematical operation on the first operand and the second operand to obtain the result; and

storing the result in the destination address in the memory.

2. (New) The method of claim 1, wherein the source address and the destination address in the memory correspond to 16 bit memory locations.

3. (New) The method of claim 1, wherein the source address and the destination address in the memory correspond to 8 bit memory locations.

4. (New) The method of claim 1, further comprising the step of passing through a sign extender the first operand in response to the source address corresponding to a memory location of less than 32 bits.

5. (New) The method of claim 4, further comprising truncating the result prior to storing in the destination address in the memory.

6. (New) The method of claim 1, wherein the instruction specifies a size of the first operand, the size being one of 8 bits, 16 bits, and 32 bits.

7. (New) The method of claim 6, wherein the size of the first operand is specified in an Operation Code within the 5 bits of the mathematical operation to be performed.

8. (New) The method of claim 1, wherein the accessing the first operand further comprises, calculating the source address in the memory from data in the 11 bits of the 32-bit instruction specifying a source address in a memory.

9. (New) The method of claim 8, wherein calculating comprises using one address mode of the group consisting of a Register + Immediate, a Register + Register Indirect, a Register + Immediate Auto-Increment, a Register Direct, and an Immediate addressing mode.

10. (New) An apparatus for performing arithmetic in a memory to memory architecture in an embedded processor, the apparatus comprising:

means for receiving a 32-bit fixed length instruction, the instruction specifying a source address in a memory using 11 bits, a source address in a register

file using 5 bits, a destination address in the memory using 11 bits, and a mathematical operation to be performed using 5 bits;

means for accessing, from the source address in the memory, a first operand on which the mathematical operation is to be performed;

means for accessing, from the source address in the register file, a second operand on which the mathematical operation is to be performed;

means for performing the mathematical operation on the first operand and the second operand to obtain the result; and

means for storing the result in the destination address in the memory.

11. (New) An embedded processor for providing connectivity in a communications system, comprising:

a 32-bit arithmetic-logic unit (ALU) comprising a first input, a second input, and an output, the ALU for performing an operation on a first 32-bit operand and a second 32-bit operand and for producing a 32-bit result, the operation specified in a 32-bit instruction fetched by the ALU;

a 32-bit register file for temporarily holding data, the 32-bit register file coupled to the first input of the ALU and communicatively coupled to the second input of the ALU for providing the first and the second 32-bit operand and coupled to the output of the ALU to receive the 32-bit result as an output from the ALU;

a memory device communicatively coupled with the second input of the ALU for providing the ALU input-data and communicatively coupled to the output

of the ALU for receiving result-data, the memory device comprising a storage location of a size of less than 32 bits;

a sign extender coupled to the memory device for expanding the input-data from the memory to 32-bit data; and

a multiplexer comprising a first and a second input and an output, the first input coupled to the sign extender for receiving the expanded 32-bit data, the second input coupled to the 32-bit register, and the output coupled to the ALU, the multiplexer for selecting between the inputs the source for providing the first 32-bit operand to the ALU.

12. (New) The embedded processor of claim 11, further comprising:

a truncator communicatively coupled to the ALU and the memory device, the truncator for converting the 32-bit result received from the ALU to a data unit of the size of the storage location in the memory device.

13. (New) The embedded processor of claim 11, wherein the multiplexer further comprises a third input communicatively coupled to an immediate.

14. (New) The embedded processor of claim 13, wherein the immediate is less than 32 bits and wherein the immediate is coupled to the sign expander for converting the immediate to an expanded 32-bit immediate, the sign expander further coupled to the third input of the multiplexer for communicating the expanded 32-bit immediate.

15. (New) The embedded processor of claim 13, wherein the 32-bit instruction comprises a 5-bit OpCode for specifying the operation, an 11-bit source address for specifying a first source location for the first 32-bit operand, a 5-bit source address for

specifying a second source location for the second 32-bit operand, and an 11-bit destination address for specifying a destination location to store the 32-bit result.

16. (New) The embedded processor of claim 15, wherein the immediate comprises a 4-bit immediate and a 7-bit immediate, the embedded processor further comprising:

a general register file having no more than 32 data registers, the general register file comprising memory addressing information;

a second multiplexer comprising a first, a second, and a third input and an output, the first input coupled to the 7-bit immediate, the second input coupled to the 4-bit immediate, and the third input coupled to the general register file, the multiplexer for selecting between the inputs to provide an output;

an address register file having no more than 8 registers, the address register file comprising addressing information and an output;

an adder coupled to the output of the second multiplexer for receiving one of the 7-bit immediate, the 4-bit immediate, and the general register file, and the adder coupled to the address register for adding the output of the second multiplexer with the output of the address register and for providing a sum; and

a third multiplexer comprising a first input, a second input, and an output, the first input coupled to the address register file, the second input coupled to the adder for receiving the sum, the third multiplexer for choosing between the address register file output and the sum thereby providing the 11-bit source address of the 32-bit instruction according to an address mode.

17. (New) The embedded processor of claim 16, wherein the address mode is one of the group consisting of a Register + Immediate, a Register + Register Indirect, a Register + Immediate Auto-Increment, a Register Direct, and an Immediate addressing mode.

18. (New) The embedded processor of claim 11, wherein the 32-bit instruction comprises a 5-bit OpCode for specifying the operation, an 11-bit source address for specifying a first source location for the first 32-bit operand, a 5-bit source address for specifying a second source location for the second 32-bit operand, and an 11-bit destination address for specifying a destination location to store the 32-bit result.

19. (New) A single-bit semaphore system in a multiple-system instruction set having an instruction for sharing a source between two or more systems, the instruction indicating the position of the bit acting as the single-bit semaphore, the single-bit semaphore system comprising:

a source having an output to provide source data to the multiple systems;

a mask comprising a first bit pattern and a second bit pattern, the first bit pattern having all logic zeroes except for a logic one at the position of the bit acting as the single-bit semaphore, and the second bit pattern having all logic ones except for a logic zero at the position of the bit acting as the single-bit semaphore; and

an ALU coupled to the mask and to the output of the source, the ALU for performing a logic OR function with the first bit pattern and the source data for setting the semaphore bit, and for performing a logic AND with the second bit pattern and the source data for clearing the semaphore bit.

20. (New) The single-bit semaphore of claim 19, wherein the source is one of an immediate, a register file, and a memory.

21. (New) A method of accessing data across boundaries of 32-bit words with a register while avoiding partial register writes, the method comprising:

accessing a first desired byte in a first 32-bit word;

storing the first desired byte in a right-most byte of the register; and

executing a shift-and-merge instruction thereby shifting register data one byte to the left and merging a second desired byte from a second 32-bit word in the right-most byte of the register.

22. (New) The method of claim 21, wherein the register is a 16-bit register.

23. (New) The method of claim 21, further comprising executing a second shift-and-merge instruction thereby shifting the register data one byte to the left and merging a third desired byte from a third 32-bit word in the right-most byte of the register.

24. (Currently amended) A method of modifying instructions stored in an instruction memory section of a multi-thread system[[s]] comprising:

assigning to each thread of a set of threads a set of time slots, wherein each time slot assigned to a thread does not overlap in time with a time slot assigned to a another thread, and wherein one instruction is performed during one time slot; and

in response to a modify instruction performed during a first time slot of a first thread, accessing an instruction stored in the instruction memory section during a

second time slot of the first thread subsequent to the first time slot of the first thread during which the modify instruction instructs to modify the instruction stored in the instruction memory.

25. (New) The method of claim 24, wherein the modify instruction is one of IREAD, IWRITE, and IERASE.

26. (New) The method of claim 24, wherein the set of threads comprises a first, a second, and a third thread and wherein the assigning comprises assigning the first thread a set of five time slots, assigning the second thread a set of 3 timeslots, and assigning the third thread a set of 2 time slots.

27. (New) The method of claim 26, wherein the set of time slots of the first, second, and third threads make up a time sequence characterized by a first time slot of the first thread, a first time slot of the second thread, a second time slot of the first thread, a first time slot of the third thread, a third time slot of the first thread, a second time slot of the second thread, a fourth time slot of the first thread, a third time slot of the second thread, a fifth time slot of the first thread, and a third time slot of the third thread.